

APPLICATION  
FOR  
UNITED STATES LETTERS PATENT

TITLE: ARRAY SEARCHING OPERATIONS

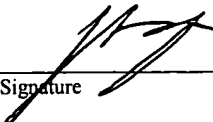
APPLICANT: CHARLES P. ROTH, RAVI KOLAGOTLA AND  
JOSE FRIDMAN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No.: EL589643016US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

September 28, 2000  
Date of Deposit  
Certificate

  
Signature

Vince Defante  
Typed Name of Person Signing



ARRAY SEARCHING OPERATIONS

BACKGROUND

This invention relates to array searching operations  
5 for a computer.

Many conventional programmable processors, such as  
digital signal processors (DSP), support a rich instruction  
set that includes numerous instructions for manipulating  
arrays of data. These operations are typically  
10 computationally intensive and can require significant  
computing time, depending upon the number of execution  
units, such as multiply-accumulate units (MACs), within the  
processor.

DESCRIPTION OF DRAWINGS

15 *Sub A1* Figure 1 is a block diagram illustrating an example of  
a pipelined programmable processor according to the  
invention.

Figure 2 is a block diagram illustrating an example  
20 execution pipeline for the programmable processor.

*Sub A2* Figure 3 is a flowchart for implementing an example  
array manipulation machine instruction according to the  
invention.

Figure 4 is a flowchart of an example routine for  
25 invoking the machine instruction.



1-15  
Q3

# DESCRIPTION

Figure 1 is a block diagram illustrating a programmable processor 2 having an execution pipeline 4 and a control unit 6. Processor 2, as explained in detail below, reduces the computational time required by array manipulation operations. In particular, processor 2 may support a machine instruction, referred to herein as the SEARCH instruction, that reduces the computational time to search an array of numbers in a pipelined processing environment.

Pipeline 4 has a number of stages for processing instructions. Each stage processes concurrently with the other stages and passes results to the next stage in pipeline 4 at each clock cycle. The final results of each instruction emerge at the end of the pipeline in rapid succession.

Control unit 6 controls the flow of instructions and data through the various stages of pipeline 4. During the processing of an instruction, for example, control unit 6 directs the various components of the pipelined to fetch and decode the instruction, perform the corresponding operation and write the results back to memory or local registers.

Figure 2 illustrates an example pipeline 4 configured according to the invention. Pipeline 4, for example, has five stages: instruction fetch (IF), decode (DEC), address calculation (AC), execute (EX) and write back (WB).



Instructions are fetched from memory, or from an instruction cache, during the IF stage by fetch unit 21 and decoded within address registers 22 during the DEC stage. At the next clock cycle, the results pass to the AC stage, where data address generators 23 calculate any memory addresses that are necessary to perform the operation.

During the EX stage, execution units 25A through 25M perform the specified operation such as, for example, adding or multiplying numbers, in parallel. Execution units 25 may contain specialized hardware for performing the operations including, for example, one or more arithmetic logic units (ALU's), floating-point units (FPU) and barrel shifters. A variety of data can be applied to execution units 25 such as the addresses generated by data address generator 23, data retrieved from data memory 18 or data retrieved from data registers 24. During the final stage (WB), the results are written back to data memory or to data registers 24.

The SEARCH instruction supported by processor 2, may allow software applications to search an array of N data elements by issuing N/M search instructions, where M is the number of data elements that can be processed in parallel by execution units 25 of pipeline 4. Note, however, that a single execution unit may be capable of executing two or more operations in parallel. For example, an execution unit may include a 32-bit ALU capable of concurrently comparing two 16-bit numbers.



Sub  
Q5

Generally, the sequence of SEARCH instructions allow the processor to process M sets of elements in parallel to identify an "extreme value", such as a maximum or a minimum, for each set. During the execution of the search instructions, processor 2 stores references to the location of the extreme value of each of the M sets of elements. Upon completion of the N/M instructions, as described in detail below, the software application analyzes the references to the extreme values for each set to quickly identify an extreme value for the array. For example, the instruction allows the software applications to quickly identify either the first or last occurrence of a maximum or minimum value. Furthermore, as explained in detail below, processor 2 implements the operation in a fashion suitable for vectorizing in a pipelined processor across the M execution units 25.

Sub  
Q6

As described above, a software application searches an array of data by issuing N/M SEARCH machine instructions to processor 2. Figure 3 is a flowchart illustrating an example mode of operation 20 for processor 2 when it receives a single SEARCH machine instruction. Process 20 is described with reference to identifying the last occurrence of a minimum value within the array of elements; however, process 20 can be easily modified to perform other functions such as identifying the first occurrence of a minimum value, the first occurrence of a maximum value or a last occurrence of a maximum value.



Sub  
A7

For exemplary purposes, process 20 is described in assuming M equals 2, i.e., processor 2 concurrently processes two sets of elements, each set having N/2 elements. However, the process is not limited as such and is readily extensible to concurrently process more than two sets of elements. In general, process 20 facilitates vectorization of the search process by fetching pairs of elements as a single data quantity and processing the element pairs through pipeline 4 in parallel, thereby reducing the total number of clock cycles necessary to identify the minimum value within the array. Although applicable to other architectures, process 20 is well suited for a pipelined processor 2 having multiple execution units in the EX stage. For each set of elements, process 20 maintains two pointer registers,  $P_{Even}$  and  $P_{Odd}$ , that store locations for the current extreme value within the corresponding set. In addition, process 20 maintains two accumulators, A0 and A1, that hold the current extreme values for the sets. The pointer registers and the accumulators, however, may readily be implemented as general-purpose data registers without departing from process 30.

Sub  
A8

Referring to Figure 3, in response to each SEARCH instruction, processor 2 fetches a pair of elements in one clock cycle as a single data quantity (21). For example, processor 2 may fetch two adjacent 16-bit values as one 32-bit quantity. Next, processor 2 compares the even element



Q8  
Cmt.

of the pair to a current minimum value for the even elements (22) and the odd element of the pair to a current minimum value for the odd elements (24).

Sub  
Q9

When a new minimum value for the even elements is detected, processor 2 updates accumulator A0 to hold the new minimum value and updates a pointer register  $P_{Even}$  to hold a pointer to corresponding data quantity within the array (23). Similarly, when a new minimum value for the odd elements has been detected, processor 2 updates accumulator A1 and a pointer register  $P_{Odd}$  (25). In this example, each pointer register  $P_{Even}$  and  $P_{Odd}$  points to the data quantity and not the individual elements, although the process is not limited as such. Processor 2 repeats the process until all of the elements within the array have been processed (26). Because processor 2 is pipelined, element pairs may be fetched until the array is processed.

The following illustrates exemplary syntax for invoking the machine instruction:

$(P_{Odd}, P_{Even}) = \text{SEARCH } R_{Data} \text{ LE, } R_{Data} = [P_{fetch\_addr}++]$

Data register  $R_{Data}$  is used as a scratch register to store each newly fetched data element pair, with the least significant word of  $R_{Data}$  holding the odd element and the most significant word of  $R_{Data}$  holding the even element. Two accumulators, A0 and A1, are implicitly used to store the actual values of the results. An additional register,  $P_{fetch\_addr}$ , is incremented when the SEARCH instruction is issued and is used as a pointer to iterate over the  $N/2$



data quantities within the array. The defined condition, such as ``less than or equal'' (LE) in the above example, controls which comparison is executed and when the pointer registers  $P_{\text{Even}}$  and  $P_{\text{Odd}}$ , as well as the accumulators A0 and A1, are updated. The ``LE'', for example, directs processor 2 to identify the last occurrence of the minimum value.

Sub  
Q10/

10 In a typical application, a programmer develops a software application or subroutine that issues the N/M search instructions, probably from within a loop construct. The programmer may write the software application in assembly language or in a high-level software language. A compiler is typically invoked to process the high-level software application and generate the appropriate machine instructions for processor 2, including the SEARCH machine instructions for searching the array of data.

Sub  
Q11/

20 Figure 4 is a flowchart of an example software routine 30 for invoking the example machine instructions illustrated above. First, the software routine 30 initializes the registers including initializing A0 and A1 and pointing  $P_{\text{Eve}}$  and  $P_{\text{Odd}}$  to the first data quantity within the array (31). In one embodiment, software routine 30 initializes a loop count register with the number of SEARCH instructions to issue (N/M). Next, routine 30 issues the SEARCH machine instruction N/M times. This can be accomplished a number of ways, such as by invoking a hardware loop construct supported by processor 2. Often,



*All*  
*cond.* however, a compiler may unroll a software loop into a sequence of identical SEARCH instructions (32).

After issuing N/M search instructions, A0 and A1 hold the last occurrence of the minimum even value and the last occurrence of the minimum odd value, respectively.  
Furthermore,  $P_{\text{Even}}$  and  $P_{\text{Odd}}$  store the locations of the two data quantities that hold the last occurrence of the minimum even value and the last occurrence of the minimum odd value.

*Sub*  
*10*  
*A12* Next, in order to identify the last occurrence of the minimum value for the entire array, routine 30 first increments  $P_{\text{Odd}}$  by a single element, such that  $P_{\text{Odd}}$  points directly at the minimum odd element (33). Routine 30 compares the accumulators A0 and A1 to determine whether the accumulators contain the same value, i.e., whether the minimum of the odd elements equals the minimum of the even elements (34). If so, the routine 30 compares the pointers to determine whether  $P_{\text{Odd}}$  is less than  $P_{\text{Even}}$  and, therefore,  $P_{\text{Odd}}$  and  $P_{\text{Even}}$  whether the minimum even value occurred earlier in the array (35). Based on the comparison, the routine determines whether to copy  $P_{\text{Odd}}$  into  $P_{\text{Even}}$  (37).

When the accumulators A0 and A1 are not the same, the routine compares A0 to A1 in order to determine which holds the minimum value (36). If A1 is less than A0 then routine 30 sets  $P_{\text{Even}}$  equal to  $P_{\text{Odd}}$ , thereby copying the pointer to the minimum value from  $P_{\text{Odd}}$  into  $P_{\text{Even}}$  (37).



At this point,  $P_{Even}$  points to the last occurrence of the minimum value for the entire array. Next, routine 30 adjusts  $P_{Even}$  to compensate for errors introduced to the pipelined architecture of processor 2 (38). For example, the comparisons described above are typically performed in the EX stage of pipeline 4 while incrementing the pointer register  $P_{fetch\_addr}$  typically occurs during the AC stage, thereby causing the  $P_{Odd}$  and  $P_{Even}$  to be incorrect by a known quantity. After adjusting  $P_{Even}$ , routine 30 returns  $P_{Even}$  as a pointer to the last occurrence of the minimum value within the array(39).

Figure 5 illustrates the operation for a single SEARCH instruction as generalized to the case where processor 2 is capable of processing M elements of the array in parallel, such as when processor 2 includes M execution units. The SEARCH instruction causes processor 2 to fetch M elements in a single fetch cycle (51). Furthermore, in this example, processor 2 maintains M pointer registers to store addresses (locations) of a corresponding extreme value for each of the M sets of elements. After fetching the M elements, processor 2 concurrently compares the M elements to a current extreme value for the respective element set, as stored in M accumulators (52). Based on the comparisons, processor 2 updates the M accumulators and the M pointer registers (53).

Figure 6 illustrates the general case where a software application issues N/M SEARCH instructions and, upon



completion of the instructions, determines the extreme value for the entire array. First, the software application initializes a loop counter, the M accumulators used to store the current extreme values for the M element sets and the M pointers used to store the locations of the extreme values (61). Next, the software application issues N/M SEARCH instructions (62). After completion of the instructions, the software application may adjust the M pointer registers to correctly reference its respective extreme value, instead of the data quantity holding the extreme value (63). After adjusting the pointer registers, the software application compares the M extreme values for the M element sets to identify an extreme value for the entire array, i.e., a maximum value or a minimum value (64). Then, the software application may use the pointer registers to determine whether more than one of the element sets have an extreme value equal to the array extreme value and, if so, determine which extreme value occurred first, or last, depending upon the desired search function (65).

Various embodiments of the invention have been described. For example, a single machine instruction has been described that searches an array of data in a manner that facilitates vectorization of the search process within a pipelined processor. The processor may be implemented in a variety of systems including general purpose computing systems, digital processing systems, laptop computers, personal digital assistants (PDA's) and cellular phones.



For example, cellular phones often maintain an array of values representing signal strength for services available 360° around the phone. In this context, the process discussed above can be readily used upon initialization of

5 the cellular phone to scan the available services and quickly select the best service. In such a system, the processor may be coupled to a memory device, such as a FLASH memory device or a static random access memory (SRAM), that stores an operating system and other software

10 applications. These and other embodiments are within the scope of the following claims.

What is claimed is: